На этом занятии мы с вами увидим, как определяются и для чего предназначены статические поля и методы класса. Начнем со статических полей.

Ранее мы с вами в курсе по Си рассматривали статические переменные и отмечали, что они создаются (размещаются в памяти) один раз и существуют на всем протяжении работы программы. Напомню, что статические переменные можно объявить либо в теле функции, тогда перед ее типом нужно прописать ключевое слово static, либо как глобальную, тогда переменная автоматически становится статической:

```language-cpp
#include <iostream>

void show_static()
{
    static int counter = 0;
    std::cout << counter++ << std::endl;
}

int global_counter = 0; // глобальная и статическая переменная

int main()
{
    show_static();
    show_static();
    show_static();

    return 0;
}
```

Аналогичные объявления переменных можно делать и в классе. Например, так:

```language-cpp
class Point {
    static unsigned counter;
    int x, y;
public:
    Point() : x(0), y(0)
        { }
};

int main()
{
```

```
    Point pt1, pt2;

    return 0;
}
```

Как правильно в классе Point воспринимать строчку:

```
static unsigned counter;
```
language-cpp

В действительности, это лишь объявление статической переменной. **Сама переменная counter при этом не создается**. Так же, как и обычные переменные x и y. **То есть, класс по-прежнему описывает лишь тип данных, шаблон, по которому следует конструировать его объекты**. И в функции main два таких объекта создаются: pt1 и pt2. Какие в итоге данные (переменные) будут содержать эти объекты? На самом деле только переменные x и y. **Статические поля объектам класса не принадлежат**. И это логично, так как статическая переменная может быть только в единственном экземпляре: она создается в момент ее инициализации и пропадает при завершении программы. По сути, объявление

```
static unsigned counter;
```
language-cpp

аналогично объявлению глобальной переменной с ключевым словом extern:

```
extern int global_counter;
```
language-cpp

Это объявление не создает саму переменную, но указывает компилятору на ее возможное наличие в этом или другом модуле программы. Поэтому если в программе попытаться вывести значение переменной counter (сделав ее предварительно публичной):

```
int main()
{
    Point pt1, pt2;
    std::cout << pt1.counter;

    return 0;
}
```
language-cpp

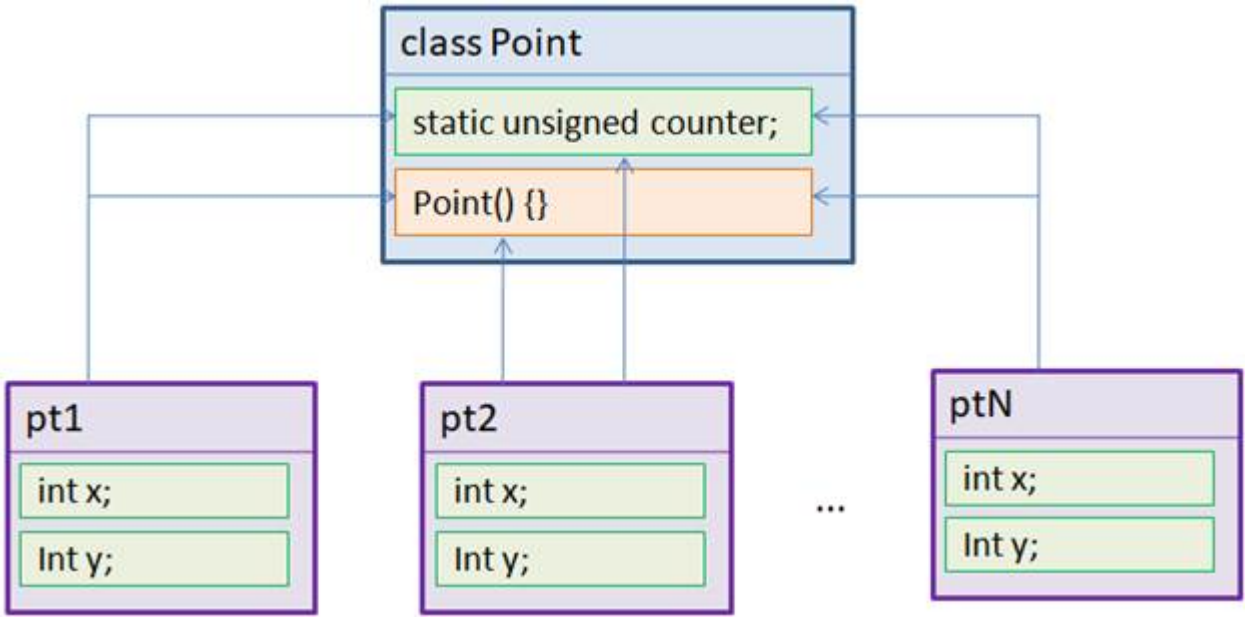
то получим ошибку:

```
undefined reference to `Point::counter'
```

Как же создать статическую переменную? Для этого ее нужно проинициализировать каким-либо значением. И делается это, как правило, за пределами описания класса следующим образом:

```cpp
unsigned Point::counter = 0;
```

В результате статическая переменная counter будет создана (размещена в памяти) и находиться в области видимости класса Point. И эта переменная будет существовать совершенно независимо от объектов класса Point. Условно это можно изобразить следующей схемой:



То есть, доступ к переменной counter возможен, как напрямую:

```cpp
Point::counter = 100;
```

так и через объекты класса:

```cpp
std::cout << pt1.counter << std::endl;
std::cout << pt2.counter << std::endl;
```

Статические методы

Помимо статических полей в классе можно объявлять и статические методы. Делается это по аналогии: перед типом статического метода прописывается ключевое слово static, например, так:

```cpp
class Point {
    static unsigned counter;
```

```
    int x, y;

public:
    Point() : x(0), y(0)
        { }
    void get_coords(int& a, int& b)
        {a = x; b = y;}
    static unsigned get_counter()
        { return counter; }
};

unsigned Point::counter = 0;
```

Обратите внимание, что, несмотря на то, что поле counter было помещено в секцию private, его инициализация за пределами этого класса вполне допустима, т.к. здесь происходит объявление статической переменной, а не обращение к ней.

В чем отличие статического метода get_counter от обычного метода, например, get_coords? По сути, только одно. **Статический метод – это обычная функция, объявленная в области видимости класса. Ее вызов не связан с конкретным объектом этого класса, а потому статический метод не имеет непосредственного доступа к переменным x, y, которые размещаются в объектах**. По этой же причине в статический метод компилятор не передает неявный указатель this, так как вызов метода не связан ни с каким объектом. Именно поэтому метод (функцию) get_counter можно вызывать напрямую, указав область видимости класса:

```language-cpp
std::cout << Point::get_counter() << std::endl;
```

Хотя, никто не запрещает вызывать этот же статический метод и через объекты класса:

```language-cpp
std::cout << pt1.get_counter() << std::endl;
```

Но, по сути, это будет эквивалентный вызов напрямую через класс.

И здесь у вас вполне может возникнуть вопрос, зачем вообще нужны эти статические поля и методы в классе? Для ответа на него в следующем занятии мы с вами разберем реализацию известного паттерна Singleton (синглтон), который, как раз, использует статические поля.